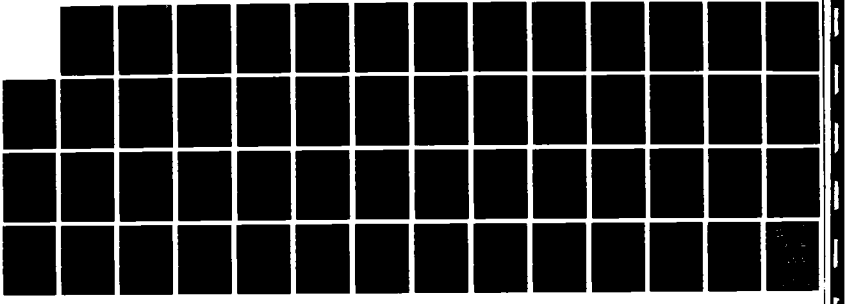
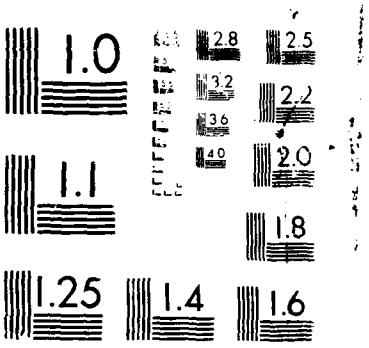
AD-A195 508 ADA (TRADE NAME) COMPILER VALIDATION SUMMARY REPORT: 1/1
ALSYS INC ALSYCOMP 0.. (U) NATIONAL COMPUTING CENTRE LTD
MANCHESTER (ENGLAND) 04 JUN 88 AVF-VSR-90502/27

UNCLASSIFIED F/G 12/5 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963

AD-A195 508

DTIC FILE COPY

E (When Data Entered)

ENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETEING FORM

12. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: Alsys Inc., AlsyCOMP_003, V3.1, Wang PC 280	5. TYPE OF REPORT & PERIOD COVERED 4 June 1987 to 4 June 1988
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) The National Computing Centre Limited Manchester, UK	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS The National Computing Centre Limited Manchester, UK	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 4 June 1987 13. NUMBER OF PAGES 53 p.
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) The National Computing Centre Limited Manchester, UK	15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, If different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

AlsyCOMP_003, V3.1, Alsys Inc., National Computing Centre Limited, Wang PC 280 under MS-DOS 3.20 (host and target). ACVC 1.8.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AVF Control Number: AVF-VSR-90502/27

Ada* COMPILER
VALIDATION SUMMARY REPORT
Alsys Inc
AlsyCOMP 003 V3.1
Wang PC 280

Completion of On-Site Testing
4 June 1987

Prepared By
The National Computing Centre Limited
Oxford Road
Manchester
M1 7ED
UK

Prepared For
Ada Joint Program Office
United States Department of Defense
Washington, D.C.
USA

---

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

Ada* Compiler Validation Summary Report:

Compiler Name: AlsyCOMP_003, V3.1

Host:
Wang PC 280
under MS-DOS 3.20

Target:
Same as host

Testing Completed 4 June 1987 using ACVC 1.8

This report has been reviewed and is approved.

The National Computing Centre Ltd
Vony Gwillim
Oxford Road
Manchester
M1 7ED

Ada Validation Office
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the AlsyCOMP_003, V3.1 using Version 1.8 of the Ada* Compiler Validation Capability (ACVC). The AlsyCOMP_003 is hosted on a Wang PC 280 operating under MS-DOS 3.20.

On-site testing was performed 1 June 1987 through 4 June 1987 at Alsys Inc., Waltham MA 02154 under the direction of the National Computing Centre (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2210 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as 170 executable tests that make use of floating-point precision exceeding that supported by the implementation were not processed. After the 2210 tests were processed, results for Class A, C, D, or E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 26 of the processed tests determined to be inapplicable; The remaining 2184 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	102	253	334	243	161	97	136	261	129	32	218	218	2184
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	14	72	86	4	0	0	3	1	1	0	0	15	196
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

---

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from characteristics of particular operating systems, hardware, or implementation strategies. All of the dependencies demonstrated during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behaviour that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any unsupported language constructs required by the Ada Standard.
- To determine that the implementation-dependent behaviour is allowed by the Ada Standard

[illegible] this compiler was conducted by NCC under the direction of [illegible] according to policies and procedures established by the Ada Validation Organisation (AVO). On-site testing was conducted from [illegible] through 4 June 198[illegible] at Alsys Inc, Waltham MA 02154.

[illegible] OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. 552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organisations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
The National Computing Centre Ltd
Oxford Road
Manchester
M1 7ED
United Kingdom

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard
Alexandria VA 22311

[illegible]PENCES

Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.

Ada Compiler Validation Capability Implementer's Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

AVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The National Computing Centre Ltd. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host	The computer on which the compiler resides.
Inapplicable test	A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn	A test found to be incorrect and not used to check conformity to test the Ada language specification. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capabilities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is passed as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimization allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values -for example, an illegal file name. A [illegible] the values used for this validation [illegible] listed in Appendix C.

[illegible] must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard either meeting the pass [illegible] given for the test or by showing that the test is [illegible]ble to the implementation. Any test that was determined to [illegible]n illegal language construct or an erroneous language [illegible] is withdrawn from the ACVC and, therefore, is not used in [illegible] compiler. The tests withdrawn at the time of validation are [illegible] Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: AlsyCOMP_003, V3.1

ACVC Version: 1.8

Certification Expiration Date:

Host Computer:

Machine :	Wang PC 280
Operating System:	MS-DOS 3.20
Memory Size:	Machine 1 and Machine 2 Base memory 640K Extended memory 4 MB Hard disk 33 MB

Target Computer:

Same as host

Communications Network: Not applicable

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behaviour of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Capacities.

  This compiler correctly processes compilations containing loop statements nested to 17 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, LONG_INTEGER and LONG_FLOAT, in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

. Array Types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/ or SYSTEM.MAX_INT.

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises no exception when the array type is declared or when the array objects are declared or sliced. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation does not raise NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are not evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

Functions

An implementation may allow the declaration of a parameterless function and an enumeration literal having the same profile in the same immediate scope, or it may reject the function declaration. If it accepts the function declarations, the use of the enumeration literal's identifier denotes the function. This implementation rejects the declarations. (See test E66001D.)

. Representation clauses.

The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses is not checked by Version 1.8 of the ACVC, they are used in testing other language features. This implementation accepts 'SIZE for tasks, and 'SMALL clauses; it rejects 'STORAGE_SIZE for tasks, and for collections. Enumeration representation clauses, including those that specify noncontiguous values, appear to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

. Pragmas.

The pragma INLINE is supported for procedures and functions. (See tests CA3004E and CA3004F.)

. Input/Output.

The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

An existing text file can be opened in OUT_FILE mode, can be created in OUT_FILE mode, and cannot be created in IN_FILE mode. (See test EE3102C.)

More than one internal file can be associated with each external file for text I/O for reading only. (See tests CE3111A..E (5 tests).)

More than one internal file can be associated with each external file for sequential I/O for reading only. (See tests CE2107A..F (6 tests).)

More than one internal file can be associated with each external file for direct I/O for reading only. (See tests CE2107A..F (6 tests).)

An external file associated with more than one internal file cannot be deleted. (See test CE2110B.)

Temporary sequential and direct files are given a name. Temporary files given names are not deleted when they are closed. (See tests CE2108A and CE2108C.)

. Generics.

Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C and BC3205D.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of IrvCOMP 503 was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 196 tests were inapplicable to this implementation, and that the 2184 applicable tests were passed by the implementation.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	69	865	1178	13	13	46	2184
Failed	0	0	0	0	0	0	0
Inapplicable	0	2	190	4	0	0	196
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	102	253	334	243	161	97	136	261	129	32	218	218	2184
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	14	72	36	4	0	0	3	1	1	0	0	15	196
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
Total	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A	C41404A	B74101B
B33203C	B45116A	C87B50A
C34018A	C48008A	C92005A
C35904A	B49006A	C940ACA
B74401A	B4A010C	CA3005A..D (4 tests)
		BC3204C

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 196 tests were inapplicable for the reasons indicated:

- C34001F and C35702A use SHORT_FLOAT which is not supported by this compiler.
- D55A03E..H (4 tests) because the compiler only processes compilations containing loop statements nested to 17 levels.

. B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

. C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

[illegible] uses the 'STORAGE_SIZE clause to specify the collection size for an access type which is not supported by this compiler. The STORAGE_SIZE clause is rejected during compilation.

[illegible] checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATION's base type. This is not the case for this implementation.

. BA2001E requires that duplicate names of subunits with a common ancestor be detected at compilation time. This compiler correctly detects the error at link time and the AVO rules that such behaviour is acceptable.

. CE2107B..E (4 tests), CE2110B, CE2111D, CE2111H, CE3111B..E (4 tests), CE3114B, and CE3115A are inapplicable because multiple internal files cannot be associated with the same external file when one file is open for writing. The proper exception is raised when multiple access is attempted.

CE2102O because mode IN_FILE is not supported for SEQUENTIAL_IO.

CE2102P because mode IN_FILE is not supported for DIRECT_IO.

. The following 170 tests make use of floating-point precision that exceeds the maximum of 15 supported by the implementation:

C24113L..Y (14 tests)
C35705L..Y (14 tests)
C35706L..Y (14 tests)
C35707L..Y (14 tests)
C35708L..Y (14 tests)
C35802L..Y (14 tests)
C45241L..Y (14 tests)
C45321L..Y (14 tests)
C45421L..Y (14 tests)
C45424L..Y (14 tests)
C45521L..Z (15 tests)
C45621L..Z (15 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of executable subsets that can be processed.

Splits were required for 14 Class B tests.

B26005A	B33001A	B37004A
B43201D	B45102A	B64012A
B62001B	B62001C	B74401F
B74401R	B91004A	B95069A
B95069B	BC3205C	

4 ADDITIONAL TESTING INFORMATION

4.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by AlsyCOMP_003 was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behaviour on all inapplicable tests.

4.2 Test Method

Testing of AlsyCOMP_003 using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of two Wang PC 280s operating under MS-DOS 3.20 which were both host and target.

A magnetic tape containing all tests was taken on site by the validation team for processing. The magnetic tape contained tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring splits during the prevalidation testing were not included in their split form on the magnetic tape.

The contents of the magnetic tape were loaded first onto a VAX 750 computer, where the required splits were performed. The processed source files were then transferred to an IBM PC AT computer via an Ethernet connection. Two Wang PC 280 machines were used to process the validation suite. Source files were then transferred to the host machines via floppy disks.

After the test files were loaded to disk, the full set of tests was compiled and linked on the Wang PC 280s and all executable tests were run. Results were written to floppy disk, moved to an IBM PC and then transferred to a VAX 750 via Ethernet. The results were then printed from the VAX 750.

The compiler was tested using command scripts provided by Alsys Inc and reviewed by the validation team. The following options were in effect for testing:

Option	Effect
CALLS=INLINED	This option allows insertion of code for subprograms inline and must be set for the pragma INLINE to be operative.

Tests were compiled, linked and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at AVF. The listings examined on-site by the validation team were also archived.

3.7.3 TEST SITE

The validation team arrived at Alsys Inc, Waltham MA 02154 on 1 June 1987 and departed after testing was completed on 4 June 1987.

APPENDIX A

COMPLIANCE STATEMENT

Alsys Inc has submitted the following compliance statement concerning the Alsys/OMP 90.

Compliance Statement

Base Configuration:

Compiler: AlsyCOMP_003 Version 3.1

Host OS: Ada* Compiler Validation Capability, Version 1.8

Host Computer:

Machine: Wang PC 280

Operating System: MS-DOS 3.20

Target Computer: Same as host

Communications Network: Not applicable

Alsys Inc has made no deliberate extensions to the Ada language standard.

Alsys Inc agrees to the public disclosure of this report.

Alsys Inc agrees to comply with the Ada trademark policy, as defined by the Ada Joint Program Office.

Arra Avakian Date: June 4, 1987

Alsys Inc
Arra Avakian
Vice President of Engineering

*Ada is registered trademark of the United States Government
(Ada Joint Program Office)

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine dependent conventions as mentioned in Chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation classes. The implementation-dependent characteristics of the AlsyCOMP 003, V3.1 are described in the following sections which discuss topics one through nine stated in Appendix F of the Ada Language Reference Manual (Ada/TEC DTD-1815A). (Implementation-specific portions of the package STANDARD are also included in this appendix. | The specification of the package STANDARD is also included in this appendix.)

```
package STANDARD is

   ...

   type INTEGER is -32_768 .. 32_767;
   type SHORT_INTEGER is range -128 .. 127;
   type LONG_INTEGER is range -2_147_483_648 .. 2_147_483_647;

   type FLOAT is digits 6 range
                     -2#1.111_1111_1111_1111_1111_1111#E+127
                  ..  2#1.111_1111_1111_1111_1111_1111#E+127;
   type LONG_FLOAT is digits 15 range
       -2#1.1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111
            _1111_1111#E1023
    ..  2#1.1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111
            _1111_1111#E1023

   type DURATION is delta 0.001 range -2_097_152.0 .. 2_097_151.999;


end STANDARD;
```

Alsys PC AT Ada Compiler

APPENDIX F

Implementation - Dependent Characteristics

Version 3.1

Alsys S.A.
29, Avenue de Versailles
78170 La Celle St. Cloud, France

Alsys, Inc.
1432 Main Street
Waltham, MA 02154, U.S.A.

Alsys Ltd.
Partridge House, Newtown Road
Henley-on-Thames,
Oxon RG9 1EN, England

Printed: June 1987

TABLE OF CONTENTS

APPENDIX F

Implementation - Dependent Characteristics

This appendix summarizes the implementation-dependent characteristics of the Alsys PC [illegible] Ada Compiler. This appendix is a required part of the *Reference Manual for the [illegible] Programming Language* (called the *RM* in this appendix).

The sections of this appendix are as follows:

1. The form, allowed places, and effect of every implementation-dependent pragma.
2. The name and the type of every implementation-dependent attribute.
3. The specification of the package SYSTEM.
4. The list of all restrictions on representation clauses.
5. The conventions used for any implementation-generated name denoting implementation-dependent components.
6. The interpretation of expressions that appear in address clauses, including those for interrupts.
7. Any restrictions on unchecked conversions.
8. Any implementation-dependent characteristics of the input-output packages.
9. Characteristics of numeric types.
10. Other implementation-dependent characteristics.
11. Compiler limitations.

The name *Alsys Runtime Executive Programs* or simply *Runtime Executive* refers to the runtime library routines provided for all Ada programs. These routines implement the Ada heap, exceptions, tasking control, and other utility functions.

General systems programming notes are given in another document, the *Application Developer's Guide* (for example, parameter passing conventions needed for interface with assembly routines).

1 Implementation-Dependent Pragmas

Ada programs can interface with subprograms written in Assembler and other languages through the use of the predefined pragma INTERFACE and the implementation-defined pragma INTERFACE_NAME.

1.1 INTERFACE

Pragma INTERFACE specifies the name of an interfaced subprogram and the name of the programming language for which parameter passing conventions will be generated. Pragma INTERFACE takes the form specified in the RM:

pragma INTERFACE (*language_name*, *subprogram_name*);

where

- *language_name* is ASSEMBLER or ADA.
- *subprogram_name* is the name used within the Ada program to refer to the interfaced subprogram.

The only two language names accepted by pragma INTERFACE are ASSEMBLER and ADA. The full implementation requirements for writing pragma INTERFACE subprograms are described in the *Application Developer's Guide*.

The language name used in the pragma INTERFACE does not have to have any relationship to the language actually used to write the interfaced subprogram. It is used only to tell the Compiler how to generate subprogram calls; that is, what kind of parameter passing techniques to use. The programmer can interface Ada programs with subroutines written in any other (compiled) language by understanding the mechanisms used for parameter passing by the Alsys PC AT Ada Compiler and the corresponding mechanisms of the chosen external language.

1.2 INTERFACE_NAME

Pragma INTERFACE_NAME associates the name of the interfaced subprogram with the external name of the interfaced subprogram. If pragma INTERFACE_NAME is not used, then the two names are assumed to be identical. This pragma takes the form:

pragma INTERFACE_NAME (*subprogram_name*, *string_literal*);

where,

- *subprogram_name* is the name used within the Ada program to refer to the interfaced subprogram.
- *string_literal* is the name by which the interfaced subprogram is referred to at link time.

The pragma INTERFACE_NAME is used to identify routines in other languages that are not named with legal Ada identifiers. Ada identifiers can only contain letters, digits, or underscores, whereas the DOS Linker allows external names to contain other characters, for example, the dollar sign ($) or commercial at sign (@). These characters can be specified in the *string_literal* argument of the pragma INTERFACE_NAME.

The pragma INTERFACE_NAME is allowed at the same places in an Ada program as the pragma INTERFACE. (Location restrictions can be found in section 13.9 of the *RM*.) However, the pragma INTERFACE_NAME must always occur after the pragma INTERFACE declaration for the interfaced subprogram.

The *string_literal* of the pragma INTERFACE_NAME is passed through unchanged to the DOS object file. (The DOS tools usually ignore the case of external identifiers. However recent versions of the PLINK86 and Microsoft Linkers have options to treat external identifiers in a case-sensitive manner.) The maximum length of the *string_literal* is 40 characters. This limit is not noted by the Compiler, but is truncated by the Binder to meet the Intel object module format standard. Certain DOS tools have smaller limits. (For example, the IBM Macro Assembler limits external identifiers to 31 characters.)

The *Runtime Executive* contains several external identifiers. All such identifiers begin with either the string "ADA@" or the string "ADAS@". Accordingly, names prefixed by "ADA@" or "ADAS@" should be avoided by the user.

Example

```
package SAMPLE_DATA is
    function SAMPLE_DEVICE  (X: INTEGER) return INTEGER;
    function PROCESS_SAMPLE (X: INTEGER) return INTEGER;
private
    pragma INTERFACE (ASSEMBLER, SAMPLE_DEVICE);
    pragma INTERFACE (ADA, PROCESS_SAMPLE);
    pragma INTERFACE_NAME (SAMPLE_DEVICE, "DEVICE@GET_SAMPLE");
end SAMPLE_DATA;
```

F.3 Other Pragmas

Pragma PRIORITY is accepted with the range of priorities running from 1 to 10 (see the definition of the predefined package SYSTEM in Section 3). Undefined priority (no pragma PRIORITY) is treated as though it were less than any defined priority value.

In addition to pragma SUPPRESS, it is possible to suppress all checks in a given compilation by the use of the Compiler option CHECKS.

2 Implementation-Dependent Attributes

P'IS_ARRAY — For a prefix P that denotes any type or subtype, this attribute yields the value TRUE if P in an array type or an array subtype; otherwise, it yields the value FALSE.

3 Specification of the package SYSTEM

```
package SYSTEM is

   --  ***************************
   --  * (1) Required Definitions. *
   --  ***************************

   type NAME is (I_80x86);
   SYSTEM_NAME : constant NAME := I_80x86;

   STORAGE_UNIT : constant := 8;
   MEMORY_SIZE  : constant := 640 * 1024;

   -- System-Dependent Named Numbers:

   MIN_INT      : constant := -(2 **31);
   MAX_INT      : constant := 2**31 - 1;
   MAX_DIGITS   : constant := 15;
   MAX_MANTISSA : constant := 31;
   FINE_DELTA   : constant := 2#1.0#E-31;

   -- For the high-resolution timer, the clock resolution is
   -- 1.0 / 1024.0.
   TICK         : constant := 1.0 / 18.2;

   -- Other System-Dependent Declarations:

   subtype PRIORITY is INTEGER range 1 .. 10;

   -- Though declared here as access to a STRING in fact this
   -- can be used anywhere an ADDRESS is required by Ada.

   -- The type ADDRESS is, in fact, implemented as an
   -- 8086/80286 segment:offset pair.

   type ADDRESS is access STRING;
   NULL_ADDRESS: constant ADDRESS := null;
```

```
--      ********************************
--      * (2) MACHINE TYPE CONVERSIONS *
--      ********************************

-- If the word / double-word operations below are used on
-- ADDRESS, then MSW yields the segment and LSW yields the
-- offset.

-- In the operations below, a BYTE_TYPE is any simple type
-- implemented on 8-bits (for example, SHORT_INTEGER), a WORD_TYPE is
-- any simple type implemented on 16-bits (for example, INTEGER), and
-- a DOUBLE_WORD_TYPE is any simple type implemented on
-- 32 bits (for example, LONG_INTEGER, FLOAT, ADDRESS).

-- Byte <==> Word conversions:

-- Get the most significant byte:
generic
      type BYTE_TYPE is private;
      type WORD_TYPE is private;
function MSB (W: WORD_TYPE) return BYTE_TYPE;

-- Get the least significant byte:
generic
      type BYTE_TYPE is private;
      type WORD_TYPE is private;
function LSB (W: WORD_TYPE) return BYTE_TYPE;

-- Compose a word from two bytes:
generic
      type BYTE_TYPE is private;
      type WORD_TYPE is private;
function WORD (MSB, LSB: BYTE_TYPE) return WORD_TYPE;


-- Word <==> Double-Word conversions:

-- Get the most significant word:
generic
      type WORD_TYPE is private;
      type DOUBLE_WORD_TYPE is private;
function MSW (W: DOUBLE_WORD_TYPE) return WORD_TYPE;

-- Get the least significant word:
generic
      type WORD_TYPE is private;
      type DOUBLE_WORD_TYPE is private;
function LSW(W: DOUBLE_WORD_TYPE) return WORD_TYPE;
```

```
        -- Compose a DATA double word from two words.
        generic
              type WORD_TYPE is private;
              -- The following type must be a data type
              -- (for example, LONG_INTEGER):
              type DATA_DOUBLE_WORD is private;
        function DOUBLE_WORD (MSW, LSW: WORD_TYPE)
                            return DATA_DOUBLE_WORD;

        -- Compose a REFERENCE double word from two words.
        generic
              type WORD_TYPE is private;
              -- The following type must be a reference type
              -- (for example, access or ADDRESS):
              type REF_DOUBLE_WORD is private;
        function REFERENCE (SEGMENT, OFFSET: WORD_TYPE)
                            return REF_DOUBLE_WORD;


        --      ****************************
        --      * (3) OPERATIONS ON ADDRESS *
        --      ****************************

        -- You can get an address via 'ADDRESS attribute or by
        -- instantiating the function REFERENCE, above, with
        -- appropriate types.

        -- Some addresses are used by the Compiler.  For example,
        -- the display is located at the low end of the DS segment,
        -- and addresses SS:0 through SS:128 hold the task control
        -- block and other information.  Writing into these areas
        -- will have unpredictable results.
        -- In real mode, the memory for DOS itself, including all the
        -- interrupt vectors is also unprotected.  Thus, any user of
        -- ASSIGN_TO_ADDRESS must be extremely careful.

        -- Note that no operations are defined to get the values of
        -- the segment registers, but if it is necessary an
        -- interfaced function can be written.

        generic
              type OBJECT is private;
        function FETCH_FROM_ADDRESS (FROM: ADDRESS) return OBJECT;

        generic
              type OBJECT is private;
        procedure ASSIGN_TO_ADDRESS (OBJ: OBJECT; TO: ADDRESS);

end SYSTEM;
```

4 Restrictions on Representation Clauses

The facilities covered in Chapter 13 of the *RM* are provided, except for the following features:

- Address clauses are not implemented.
- There is no bit implementation for any of the representation clauses.
- The Record Clause is not allowed for a derived record type.
- Change of representation for RECORD type is not implemented.
- The Enumeration Clause is not allowed if there is a range constraint on the parent subtype.
- For the length clause:
  - Size specification: T'SIZE is not implemented for types declared in a generic unit.
  - Specification of storage for a task activation: T'STORAGE_SIZE is not implemented when T is a task type.
  - Specification of *small* for a fixed point type: T'SMALL is restricted to a power of 2, and the absolute value of the exponent must be less than 31.
- Machine code insertions are not implemented; use instead pragma INTERFACE to ASSEMBLER to write assembly language routines.

5 Conventions for Implementation-Generated Names

The Alsys PC AT Ada Compiler may add fields to record objects and have descriptors in memory for record or array objects. These fields are not accessible to the user through any implementation-generated name or attribute.

The following predefined packages are reserved to Alsys and cannot be recompiled in Version 3.1:

```
ALSYS_ADA_RUNTIME
ALSYS_BASIC_IO
ALSYS_BASIC_DIRECT_IO
ALSYS_BASIC_SEQUENTIAL_IO
```

6 Address Clauses

This version of the Alsys PC AT Ada Compiler does not support address clauses. Support is provided for Ada interrupt entries; please see Chapter 6 of the *Application Developer's Guide* for details.

7 Restrictions on Unchecked Conversions

Unchecked conversions are allowed between any types. It is the programmer's responsibility to determine if the desired effect is achieved

8 Input-Output Packages

The *RM* defines the predefined input-output packages SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO, and describes how to use the facilities available within these packages. The *RM* also defines the package IO_EXCEPTIONS, which specifies the exceptions that can be raised by the predefined input-output packages.

In addition the *RM* outlines the package LOW_LEVEL_IO, which is concerned with low-level machine-dependent input-output, such as would possibly be used to write device drivers or access device registers. LOW_LEVEL_IO has not been implemented. The use of interfaced subprograms is recommended as an alternative.

8.1 Correspondence between External Files and DOS Files

Ada input-output is defined in terms of external files. Data is read from and written to external files. Each external file is implemented as a standard DOS file, including the use of STANDARD_INPUT and STANDARD_OUTPUT.

The name of an external file can be either

- the null string
- a DOS filename
- a DOS special file or device name (for example, CON and PRN)

If the name is a null string, the associated external file is a temporary file and will cease to exist when the program is terminated. The file will be placed in the current directory and its name will be chosen by DOS.

If the name is a DOS filename, the filename will be interpreted according to standard DOS conventions (that is, relative to the current directory). The exception NAME_ERROR is raised if the name part of the filename has more than 8 characters or if the extension part has more than 3 characters.

If an existing DOS file is specified to the CREATE procedure, the contents of the file will be deleted before writing to the file.

If a non-existing directory is specified in a file path name to CREATE, the directory will not be created, and the exception NAME_ERROR is raised.

8.2 Error Handling

DOS errors are translated into Ada exceptions, as defined in the *RM* by package IO_EXCEPTIONS. In particular, DEVICE_ERROR is raised in cases of drive not ready, unknown media, disk full or hardware errors on the disk (such as read or write fault).

8.3 Use of FORM Parameter

The only accepted value of the FORM parameter of the various CREATE and OPEN procedures is the null string. The functions FORM of TEXT_IO and (any instantiation of) DIRECT_IO and SEQUENTIAL_IO always return the null string. USE_ERROR is raised if a non-null FORM string is passed to any CREATE or OPEN procedure.

8.4 Sequential Files

For sequential access the file is viewed as a sequence of values that are transferred in the order of their appearance (as produced by the program or run-time environment). This is sometimes called a *stream* file in other operating systems. Each object in a sequential file has the same binary representation as the Ada object in the executable program.

8.5 Direct Files

For direct access the file is viewed as a set of elements occupying consecutive positions in a linear order. The position of an element in a direct file is specified by its index, which is an integer of subtype POSITIVE_COUNT.

DIRECT_IO only allows input-output for constrained types. If DIRECT_IO is instantiated for an unconstrained type, all calls to CREATE or OPEN will raise USE_ERROR. Each object in a direct file will have the same binary representation as the Ada object in the executable program. All elements within the file will have the same length.

8.6 Text Files

Text files are used for the input and output of information in ASCII character form. Each text file is a sequence of characters grouped into lines, and lines are grouped into a sequence of pages.

All text file column numbers, line numbers, and page numbers are values of the subtype POSITIVE_COUNT.

Note that due to the definitions of line terminator, page terminator, and file terminator in the *RM*, and the method used to mark the end of file under DOS, some ASCII files do not represent well-formed TEXT_IO files.

A text file is buffered by the *Runtime Executive* unless

- it names a device (for example, CON or PRN).
- it is STANDARD_INPUT or STANDARD_OUTPUT and has not been redirected.

If not redirected, prompts written to STANDARD_OUTPUT with the procedure PUT will appear before (or when) a GET (or GET_LINE) occurs.

The functions END_OF_PAGE and END_OF_FILE always return FALSE when the file is a device, which includes the use of the file CON, and STANDARD_INPUT when it is not redirected. Programs which would like to check for end of file when the file may be a device should handle the exception END_ERROR instead, as in the following example:

Example

```
begin
    loop
        -- Display the prompt:
        TEXT_IO.PUT ("--> ");
        -- Read the next line:
        TEXT_IO.GET_LINE (COMMAND, LAST);
        -- Now do something with COMMAND (1 .. LAST)
    end loop;
exception
    when TEXT_IO.END_ERROR =>
        null;
end;
```

END_ERROR is raised for STANDARD_INPUT when ^Z (ASCII.SUB) is entered at the keyboard.

8.7 Access Protection of External Files

Standard DOS does not provide any access protection for external files. In a network environment, access protection is dependent on the network operating system.

8.8 The Need to Close a File Explicitly

The *Runtime Executive* will flush all buffers and close all open files when the program is terminated, either normally or through some exception.

However, the *RM* does not define what happens when a program terminates without closing all the opened files. Thus a program which depends on this feature of the *Runtime Executive* might have problems when ported to another system.

8.9 Limitation on the procedure RESET

An internal file opened for input cannot be RESET for output. However, an internal file opened for output can be RESET for input, and can subsequently be RESET back to output.

8.10 Sharing of External Files and Tasking Issues

Several internal files can be associated with the same external file only if all the internal files are opened with mode IN_MODE. However, if a file is opened with mode OUT_MODE and then changed to IN_MODE with the RESET procedure, it cannot be shared.

Care should be taken when performing multiple input-output operations on an external file during tasking because the order of calls to the I/O primitives is unpredictable. For example, two strings output by TEXT_IO.PUT_LINE in two different tasks may appear in the output file with interleaved characters. Synchronization of I/O in cases such as this is the user's responsibility.

The TEXT_IO files STANDARD_INPUT and STANDARD_OUTPUT are shared by all tasks of an Ada program.

If TEXT_IO.STANDARD_INPUT is not redirected, it will not block a program on input. All tasks not waiting for input will continue running.

9 Characteristics of Numeric Types

9.1 Integer Types

The ranges of values for integer types declared in package STANDARD are as follows:

```
SHORT_INTEGER     -128 .. 127                   -- 2**7 - 1

INTEGER           -32768 .. 32767               -- 2**15 - 1

LONG_INTEGER      -2147483648 .. 2147483647     -- 2**31 - 1
```

For the packages DIRECT_IO and TEXT_IO, the range of values for types COUNT and POSITIVE_COUNT are as follows:

```
COUNT             0 .. 2147483647               -- 2**31 - 1

POSITIVE_COUNT    1 .. 2147483647               -- 2**31 - 1
```

For the package TEXT_IO, the range of values for the type FIELD is as follows:

```
FIELD           0 .. 255                    -   2**8 - 1
```

9.2 Floating Point Type Attributes

	FLOAT	LONG_FLOAT
DIGITS	6	15
MANTISSA	21	51
EMAX	84	204
EPSILON	9.53674E-07	8.88178E-16
LARGE	1.93428E+25	2.57110E+61
SAFE_EMAX	125	1021
SAFE_SMALL	1.17549E-38	2.22507E-308
SAFE_LARGE	4.25353E+37	2.24712E+307
FIRST	-3.40282E+38	-1.79769E+308
LAST	3.40282E+38	1.79769E+308
MACHINE_RADIX	2	2
MACHINE_MANTISSA	24	53
MACHINE_EMAX	128	1024
MACHINE_EMIN	-125	-1021
MACHINE_ROUNDS	true	true
MACHINE_OVERFLOWS	false	false
SIZE	32	64

9.3 Attributes of Type DURATION

```
DURATION'DELTA          0.001

DURATION'SMALL          0.0009765625 (= 2**(-10))

DURATION'FIRST          -2097152.0
```

DURATION'LAST	2097151.999
DURATION'LARGE	same as DURATION'LAST

10 Other Implementation-Dependent Characteristics

10.1 Use of the Floating-Point Coprocessor (8087 or 80287)

The Alsys PC AT Ada Compiler generates instructions to use the floating point coprocessor for all floating point operations (but, of course, not for operations involving only *universal_real*).

A floating point coprocessor, 8087 or 80287, is required for the execution of programs that use arithmetic on floating point values. The coprocessor is needed if the FLOAT_IO or FIXED_IO packages of TEXT_IO are used.

For the AT, the *Runtime Executive* will detect the absence of the floating point coprocessor if it is required by a program and will raise NUMERIC_ERROR. The user is warned to not run a floating point program on an XT without an 8087. The result of doing so is that the XT will wait indefinitely until a power on reset.

10.2 Characteristics of the Heap

UNCHECKED_DEALLOCATION is implemented for all Ada access objects except access objects to tasks. Use of UNCHECKED_DEALLOCATION on a task object will lead to unpredictable results.

There is no management of collections (via the 'STORAGE_SIZE representation clause on access type) although *small* objects are managed more efficiently than *normal* heap objects.

All objects whose visibility is linked to a subprogram, task body, or block have their storage reclaimed at exit.

The maximum size of the heap is limited only by available memory. User programs making use of the Protected Mode feature have a heap which includes all available memory below the 640K limit plus the size of the heap file which is in the extended memory RAM disk.

All objects created by allocators go into the heap. Also, portions of the *Runtime Executive* representation of task objects, including the task stacks, are allocated in the heap.

10.3 Characteristics of Tasks

The default task stack size is 1K bytes (32K bytes for the environment task), but by using the Binder option STACK.TASK or the SETOPT program, the size for all task stacks in a program may be set to a size from 1K bytes to 64K bytes.

Normal priority rules are followed for preemption, where PRIORITY values are in the range 1 .. 10. A task with *undefined* priority (no pragma PRIORITY) is considered to be lower than priority 1.

The minimum timeable delay is

- 1.0/18.2 seconds on a PC XT, or
- 1.0/1024.0 seconds on a PC AT when the high-resolution timer is in effect through the TIMER = FAST option of the BIND command.

The maximum number of active tasks is restricted only by memory usage.

The accepter of a rendezvous executes the accept body code in its own stack. Rendezvous with an empty accept body (for synchronization) does not cause a context switch.

The main program waits for completion of all tasks dependent upon library packages before terminating.

Abnormal completion of an aborted task takes place immediately, except when the abnormal task is the caller of an entry that is engaged in a rendezvous, or if it is in the process of activating some tasks. Any such task becomes abnormally completed as soon as the state in question is exited.

The message

```
GLOBAL BLOCKING SITUATION DETECTED
```

is printed to STANDARD_OUTPUT when the *Runtime Executive* detects that no further progress is possible for any task in the program. The execution of the program is then abandoned.

10.4 Definition of a Main Subprogram

A library unit can be used as a main subprogram if and only if it is a procedure that is not generic and that has no formal parameters.

10.5 Ordering of Compilation Units

The Alsys PC AT Ada Compiler imposes no additional ordering constraints on compilations beyond those required by the language. However, if a generic unit is instantiated during a compilation, its body must be compiled prior to the completion of that compilation.

11 Limitations

11.1 Compiler Limitations

- The maximum identifier length is 255 characters.
- The maximum line length is 255 characters.
- The maximum number of unique identifiers per compilation unit is 2500.
- The maximum number of compilation units in a library is 1000.

11.2 Hardware Related Limitations

- The maximum size of the generated code for a single compilation unit is 65535 bytes.
- The maximum size of a single array or record object is 65522 bytes. The maximum size of a static record is 4096 bytes.
- The maximum size of a single stack frame is 32766 bytes, including the data for inner package subunits "unnested" to the parent frame.
- The maximum amount of data in the global data area is 65535 bytes, including compiler generated data that goes into the GDA (about 8 bytes per compilation unit plus 4 bytes per externally visible subprogram).
- The maximum amount of data in the heap is limited only by available memory.

11.3 Runtime Executive Limitations

- Task stacks are only allocated in low memory (that is, below 640k).

INDEX

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

NAME AND MEANING	VALUE
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	A....A1 | | 254 characters
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	A....A2 | | 254 characters
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	A....A3A....A | | | | 127 127 characters
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	A....A4A....A | | | | 127 127 characters
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that is is the size of the maximum line length.	0....0298 | | 252 characters
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value of 690.0, and has enough leading zeroes to be the size of the maximum line length.	0....069.0E1 | | 249 characters

NAME AND MEANING	VALUE
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	235 blanks
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz !$%?@[\]^'{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST	255
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long if no invalid characters exist.	X}]!@ $^&~Y
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long if no wild card characters exists.	XYZ*
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in in the range of DURATION.	2_097_151.01
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	10_000_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	BAD-CHARACTER*^

NAME AND MEANING	VALUE
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	MUCH-TOO-LONG-NAME-FOR-A-FILE
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-32768
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	32767
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-10_000_000.0
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	15
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	255
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	LONG_LONG_INTEGER

NAME AND MEANING	VALUE
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	8#777777777776#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 19 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- B22114A: An unterminated string literal occurs at line 62.
- B[illegible]: The reserved word "IS" is misspelled at line 45.
- C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.
- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.
- B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.
- C41404A: The values of 'LAST and 'LENGTH are incorrect in the *if* statements from line 74 to the end of the test.
- B45116A: ARRPRIBL 1 and ARRPRIBL 2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.
- C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.
- B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and *end case;* is missing from line 42.
- B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

. B74101B: The begin at line 9 causes a declarative part to be treated as a sequence of statements.

. C87B50A: The call of "/=" at line 31 requires a use clause for package A.

C92005A: The "/=" for type PACK.BIG_INT at line 40 is not visible without a use clause for the package PACK.

[illegible]ACA: The assumption that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program, is erroneous.

[illegible]A..D: No valid elaboration order exists for these tests.
[illegible]sts)

[illegible]C: The body of BC3204C0 [illegible] missing.

END

DATE

FILMED

8-88

DTIC